Com Sci 31 – Winter 2019

Project 7 Report

Name – Suyash Kumar

UID – 605108040

Discussion – 1C

# Obstacles Overcame

* Understanding what the specs of the project demanded was a tedious task, especially due to numerous classes and even greater number of functions. It was a tricky task to see how everything fit together to make the game work.
* Completing this project required a good understanding of aggregation, as the classes involved several “has-a” relationships amongst them.
* It was important to understand how data was separated among the four classes – Die handled the randomness of a dice roll, Player handed nuances related to maintaining a Player’s statistics, Board simply kept track of both players’ statistics as well as other details like status of the game, most recent rolled value, and winner (if any), and Pig combined functionality of all these classes to make the game work. The main supplied was completing the game interface and any remaining logic handling which the classes did not handle.
* Devising test cases in an organized manner was a rather tough task considering the extremely large number of functions. Nonetheless, I tried to organize the test cases according to classes, test for legal and illegal behavior wherever possible, and make the list as exhaustive as possible.

# Test Cases

Test cases for the following project may be categorized for each class, thoroughly testing every function in each class for various kinds of input.

Test Cases for Die Class

Test cases for Die class will ensure that the roll() function will return a random value that lies between 1 and mSides (default value 6).

Die d;

for (int i = 1; i <= 100; i++ )

{

d.roll();

int value = d.getValue();

assert( value >=1 && value <= 6 ); *// getValue() must return something between 1 and 6, both inclusive.*

}

Test Cases for Player Class

Test cases for Player class ensure proper functioning of roll(), endTurn(), getScore(), and getTotal() functions. The test cases represent various combinations in which these functions can be called together, the purpose of these combinations being pronounced in the comments.

* roll() should return the rolled value (random or forced) and appropriately update score, unless it returns 1, when mScore should to set to 0.
* endTurn() should update total and set score to 0.
* getScore() and getTotal() – return current values of mScore and mTotal.

Player p;

*// ensuring that mScore and mTotal are initialized to 0.*

assert(p.getScore( ) == 0 );

assert(p.getTotal( ) == 0 );

*// mScore is updated everytime the roll() function is called, except when roll() returns 1. mTotal isn't updated until endTurn() function is called.*

assert( p.roll( 6 ) == 6 );

assert( p.getScore() == 6 );

assert( p.roll( 5 ) == 5 );

assert( p.getScore() == 11 );

*// after endTurn() function is called, mTotal should be updated and mScore should be set to 0.*

p.endTurn();

assert( p.getScore() == 0 );

assert( p.getTotal() == 11 );

*// as we call roll() function again after calling endTurn(), mScore begins getting updated from 0.*

assert( p.roll( 4 ) == 4 );

assert( p.getScore() == 4 );

assert( p.roll( 5 ) == 5 );

assert( p.getScore() == 9 );

assert( p.roll( 6 ) == 6 );

assert( p.getScore() == 15 );

*// again, when endTurn() is called, mTotal is updated from its previous value, and mScore is set to 0.*

p.endTurn();

assert( p.getScore() == 0 );

assert( p.getTotal() == 26 );

*// when we call roll(), mScore starts getting updated from 0 again.*

assert( p.roll( 4 ) == 4 );

assert( p.getScore() == 4 );

assert( p.roll( 5 ) == 5 );

assert( p.getScore() == 9 );

*// as soon as roll() function returns 1, mScore is set to 0 and no changes are made to mTotal. This is like endTurn() with no changes to mTotal.*

assert( p.roll( 1 ) == 1 );

assert( p.getScore() == 0 );

assert( p.getTotal() == 26 );

Test Cases for Board Class

The test cases for the Board Class have been classified into two categories – one for handling legal behavior and the other for handling illegal behavior (both of which are appropriately explained by comments). These test cases thoroughly test all the functions which are part of the Board class by calling them in a number of possible combinations – the purpose of each combination being pronounced in the comments.

Legal Behavior

Board board; *// using board object to test all legal operations.*

*// ensuring that all the member variables related to Board class are appropriately initialized.*

assert( board.getComputerTotal() == 0 );

assert( board.getHumanTotal() == 0);

assert( board.getRolledValue() == 0);

assert( board.getComputerScore() == 0 );

assert( board.getHumanScore() == 0);

assert( board.isHumanWinner() == false );

assert( board.isHumanTurn() == true ); *// first turn must always go to the human.*

assert( board.isGameOver() == false );

*// one can make changes to mComputerScore and mComputerTotal only if mIsHumanTurn is false.*

board.setComputerTurn(); *// sets mHumanTurn to false.*

board.setComputerTotal(1);

board.setComputerScore(2);

*// similarly, changes to mHumanScore and mHumanTotal are allowed only if mHumanTurn is true.*

board.setHumanTurn(); *// sets mHumanTurn to true.*

board.setHumanScore(3);

board.setHumanTotal(4);

board.setRolledValue(7);

*// using asserts on accessor functions for testing whether mutator functions appropriately changed member variables.*

assert( board.getComputerTotal() == 1 );

assert( board.getHumanTotal() == 4);

assert( board.getRolledValue() == 7);

assert( board.getComputerScore() == 2 );

assert( board.getHumanScore() == 3);

assert( board.isHumanWinner() == false );

assert( board.isHumanTurn() == true );

assert( board.isGameOver() == false );

Illegal behavior

Board b1; *// using object b1 to ensure that Board class can handle all kinds of illegal operations.*

*// legally setting mComputerTotal and mComputerScore to some values before attempting to alter them illegally*

b1.setComputerTurn();

b1.setComputerTotal(21);

b1.setComputerScore(22);

*// computer trying to make changes in a human turn*

b1.setHumanTurn();

b1.setComputerTotal( 50 );

b1.setComputerScore( 50 );

assert( b1.getComputerTotal() == 21 ); *// no changes were made to mComputerTotal*

assert( b1.getComputerScore() == 22 ); *// no changes were made to mComputerScore*

*// legally setting mHumanTotal and mHumanScore (since mIsHumanTurn is already true) before attempting to alter them illegally*

b1.setHumanScore(23);

b1.setHumanTotal(30);

b1.markHumanAsWinner();

*// human trying to make changes in a computer turn*

b1.setComputerTurn();

b1.setHumanScore(100);

b1.setHumanTotal(150);

assert( b1.getHumanScore() == 23 );

assert( b1.getHumanTotal() == 30 );

*// computer trying to make changes after game is over*

b1.setGameOver(true);

b1.setComputerTurn();

b1.setComputerScore(10000);

b1.setComputerTotal(15000);

assert( b1.getComputerTotal() == 21 ); *// no changes were made to mComputerTotal*

assert( b1.getComputerScore() == 22 ); *// no changes were made to mComputerScore*

assert( b1.isHumanWinner() == true ); *// human still remains the winner*

*// human trying to make changes after game is over, to no avail.*

b1.setHumanTurn();

b1.setHumanScore(10);

b1.setHumanTotal(20);

assert( b1.getHumanScore() == 23 );

assert( b1.getHumanTotal() == 30 );

Test Cases for Pig Class

Since the Pig Class is used to simulate games, and there are several possible combinations of moves which could legally be operated, it is not practical to have an exhaustive list of test cases. Thus, the strategy for testing the Pig Class is to have various Pig objects simulating certain gameplays, each of which individually cover various scenarios of moves that could be made by using the functions of the Pig Class. This will cover handling of illegal cases too (like a player trying to alter the total after the game has ended).

First Game

Pig game1;

assert( game1.isGameOver() == false ); *// game has not yet started*

assert( game1.determineGameOutcome() == Pig::GAMEOUTCOME::GAMENOTOVER );

*// setting human's turn and playing.*

game1.humanPlay( 5 );

game1.humanEndTurn( );

*// human played 5 and ended turn, human's total is 5.*

*// setting computer's turn and playing.*

game1.computerPlay( 5 );

game1.computerEndTurn( );

*// computer played 5 and ended turn, computer's total is 5.*

assert( game1.isGameOver() == false ); *// game still in progress.*

assert( game1.determineGameOutcome() == Pig::GAMEOUTCOME::GAMENOTOVER );

*// setting human's turn and playing 15.*

game1.humanPlay( 5 );

game1.humanPlay( 5 );

game1.humanPlay( 5 );

game1.humanEndTurn( );

*// human played 15 and ended turn, human's total is 20.*

assert( game1.isGameOver() == false ); *// game still in progress.*

assert( game1.determineGameOutcome() == Pig::GAMEOUTCOME::GAMENOTOVER );

*// setting computer's turn and rolling 15.*

game1.computerPlay( 5 );

game1.computerPlay( 5 );

game1.computerPlay( 5 );

game1.computerEndTurn();

*// computer played 15 and ended turn, computer's total is 20.*

assert( game1.isGameOver() == false ); *// game still in progress.*

assert( game1.determineGameOutcome() == Pig::GAMEOUTCOME::GAMENOTOVER );

*// setting human turn and playing 11.*

game1.humanPlay( 5 );

game1.humanPlay( 6 );

game1.humanEndTurn( );

*// human played 11 and ended turn, human's total is 31. Game must end now and human must be declared winner.*

assert( game1.isGameOver() == true ); *// game has now ended.*

assert( game1.determineGameOutcome() == Pig::GAMEOUTCOME::HUMANWONGAME ); *// human is the winner.*

*// once the game is over, scores can't be changed*

game1.computerPlay( 6 );

game1.computerPlay( 6 );

game1.computerPlay( 6 );

game1.computerEndTurn( );

assert( game1.isGameOver() == true ); *// the game is still over.*

assert( game1.determineGameOutcome() == Pig::GAMEOUTCOME::HUMANWONGAME ); *// the human is still the winner.*

Second Game

Pig game2;

assert( game2.isGameOver() == false ); *// game has not yet started*

assert( game2.determineGameOutcome() == Pig::GAMEOUTCOME::GAMENOTOVER );

*// setting human's turn and playing.*

game2.humanPlay( 53 ); *// we should accept whatever value is coerced, legal or not.*

game2.humanEndTurn( );

*// human played 53 and ended turn, human's total is 53. Game must end now and human must be declared winner.*

assert( game2.isGameOver() == true ); *// game has now ended.*

assert( game2.determineGameOutcome() == Pig::GAMEOUTCOME::HUMANWONGAME ); *// human is the winner.*

Third Game

Pig game3;

assert( game3.isGameOver() == false ); *// game has not yet started*

assert( game3.determineGameOutcome() == Pig::GAMEOUTCOME::GAMENOTOVER );

*// human starts playing and rolls a score of 16.*

game3.humanPlay(5);

game3.humanPlay(6);

game3.humanPlay(5);

game3.humanEndTurn();

*// human ends turn and updates total to 16.*

*// computer starts playing and rolls a score of 12.*

game3.computerPlay(5);

game3.computerPlay(5);

game3.computerPlay(2);

game3.computerEndTurn();

*// computer ends turn and updates total to 12.*

*// human starts playing and ends up rolling 1.*

game3.humanPlay(3);

game3.humanPlay(1);

game3.humanEndTurn();

*// due to rolling 1, human's score is dropped to 0, and total stays 16.*

*// computer starts playing and rolls a score of 20.*

game3.computerPlay(5);

game3.computerPlay(5);

game3.computerPlay(5);

game3.computerPlay(5);

game3.computerEndTurn();

*// computer ends turn and updates total to 32 and wins.*

assert( game3.isGameOver() == true ); *// game has now ended.*

assert( game3.determineGameOutcome() == Pig::GAMEOUTCOME::COMPUTERWONGAME ); *// computer is the winner.*

Fourth Game

Pig game4;

assert( game4.isGameOver() == false ); *// game has not yet started*

assert( game4.determineGameOutcome() == Pig::GAMEOUTCOME::GAMENOTOVER );

*// computer starts playing and rolls a score of 5. Computer starts playing here first, for the sake of uniqueness.*

game4.computerPlay(2);

game4.computerPlay(5);

game4.computerEndTurn();

*// computer ends turn and updates total to 5.*

*// human starts playing and rolls a score of 10.*

game4.humanPlay(2);

game4.humanPlay(3);

game4.humanPlay(5);

game4.humanEndTurn();

*// human ends turn and updates total to 10.*

*// although it is illegal, human plays again and rolls a score of 3.*

game4.humanPlay(3);

game4.humanEndTurn();

*// human ends turn and updates total to 13.*

*// computer plays and rolls a score of 15 in one go, although it is illegal.*

game4.computerPlay(15);

game4.computerEndTurn();

*// computer ends turn and updates total to 20.*

*// human plays and ends up rolling a 1.*

game4.humanPlay(6);

game4.humanPlay(4);

game4.humanPlay(1);

game4.humanEndTurn();

*// due to rolling 1, human's score is dropped to 0 and total remains 13.*

*// computer plays and rolls a 15 and wins.*

game4.computerPlay(5);

game4.computerPlay(5);

game4.computerPlay(5);

game4.computerEndTurn();

*// computer ends turn and updates total to 35 and wins.*

assert( game4.isGameOver() == true ); *// game has now ended.*

assert( game4.determineGameOutcome() == Pig::GAMEOUTCOME::COMPUTERWONGAME ); *// computer is the winner.*

*// although the human tries to update their total, that doesn't happen since the game has ended.*

game4.humanPlay(6);

game4.humanPlay(6);

game4.humanPlay(6);

game4.humanEndTurn();

assert( game4.isGameOver() == true ); *// the game is still over.*

assert( game4.determineGameOutcome() == Pig::GAMEOUTCOME::COMPUTERWONGAME ); *// computer is still the winner.*